*# AIM*

*# ggplot2 boxplot with the possibility to add background or foreground dots*

*# for ggplot2 specifications, see: https://ggplot2.tidyverse.org/articles/ggplot2-specs.html*

*# WARNINGS*

*# rows containing NA in data1[, c(y, categ)] will be removed before processing, with a warning (see below)*

*# hinges are not computed like in the classical boxplot() function of R.*

*# to have a single boxplot, create a factor column with a single class and specify the name of this column in categ argument as unique element (no categ2 in categ argument). For a single set of grouped boxplot, create a factor column with a single class and specify this column in categ argument as first element (categ1). See categ below*

*# with several single boxplots (categ argument with only one element), box.width argument (i.e., width argument of ggplot2::geom\_boxplot()) defines each box width. The box.width argument also defines the space between boxs by using (1 - box.width). In addition, xmin and xmax of the fun\_gg\_boxplot() output report the box boundaries (around x-axis unit 1, 2, 3, etc., for each box)*

*# with several sets of grouped boxs (categ argument with two elements), box.width argument defines each set of grouped box width. The box.width argument also defines the space between set of grouped boxs by using (1 - box.width). In addition, xmin and xmax of the fun\_gg\_boxplot() output report the box boundaries (around x-axis unit 1, 2, 3, etc., for each set of grouped box)*

*# ARGUMENTS*

*# data1: a dataframe containing one column of values (see y argument below) and one or two columns of categories (see categ argument below). Duplicated column names not allowed*

*# y: character string of the data1 column name for y-axis (containing numeric values). Numeric values will be averaged by categ to generate the boxs and will also be used to plot the dots*

*# categ: vector of character strings of the data1 column name for categories (column of characters or factor). Must either be one or two column names. If a single column name (further refered to as categ1), then one box per class of categ1. If two column names (further refered to as categ1 and categ2), then one box per class of categ2, which form a group of boxs in each class of categ1. BEWARE, categ1 (and categ2 if it exists) must have a single value of y per class of categ1 (and categ2). To have a single box, create a factor column with a single class and specify the name of this column in categ argument as unique element (no categ2 in categ argument). For a single set of grouped boxs, create a factor column with a single class and specify this column in categ argument as first element (categ1)*

*# categ.class.order: list indicating the order of the classes of categ1 and categ2 represented on the boxplot (the first compartment for categ1 and and the second for categ2). If categ.class.order = NULL, classes are represented according to the alphabetical order. Some compartment can be NULL and other not*

*# categ.legend.name: character string of the legend title for categ2. If categ.legend.name = NULL, then categ.legend.name <- categ1 if only categ1 is present and categ.legend.name <- categ2 if categ1 and categ2 are present. Write "" if no legend required*

*# categ.color: vector of character color string for box frame. If categ.color = NULL, default colors of ggplot2, whatever categ1 and categ2. If categ.color is non null and only categ1 in categ argument, categ.color can be either: (1) a single color string (all the boxs will have this color, whatever the classes of categ1), (2) a vector of string colors, one for each class of categ1 (each color will be associated according to categ.class.order of categ1), (3) a vector or factor of string colors, like if it was one of the column of data1 data frame (beware: a single color per class of categ1 and a single class of categ1 per color must be respected). Integers are also accepted instead of character strings, as long as above rules about length are respected. Integers will be processed by fun\_gg\_palette() using the max integer value among all the integers in categ.color. If categ.color is non null and categ1 and categ2 specified, all the rules described above will apply to categ2 instead of categ1 (colors will be determined for boxs inside a group of boxs)*

*# box.fill: logical. Fill the box? If TRUE, the categ.color argument will be used to generate filled boxplot (the box frames being black) as well as filled outlier dots (the dot border being controled by the dot.border.color argument) and if all the dots are plotted (argument dot.color other than NULL), they will be over the boxes. If FALSE, the categ.color argument will be used to color the box frames and the outlier dot borders, and if all the dots are plotted, they will be beneath the boxes*

*# box.width: numeric value (from 0 to 1) of the box or set of grouped box width (see warnings above)*

*# box.space: numeric value (from 0 to 1) indicating the box separation in grouped boxes. 0 means no space and 1 means boxes shrinked to a vertical line. Ignored if no grouped boxes*

*# box.line.size: numeric value of line size of boxes and whiskers (in mm)*

*# box.notch: logical. Notched boxplot? It TRUE, display notched boxplot, the notches corresponding approximately to the 95% confidence interval of the median (the notch interval is exactly 1.58 x Inter Quartile Range (IQR) / sqrt(n), with n the number of values that made the box). If notch intervals between two boxes do not overlap, it can be interpreted as significant median differences*

*# box.alpha: numeric value (from 0 to 1) of box transparency (full transparent to full opaque, respectively). BEWARE: work only for the fill of boxplots, not for the frame. See https://github.com/tidyverse/ggplot2/issues/252*

*# box.mean: logical. Add mean value? It TRUE, a losange dot, additional to the solid median bar and corresponding to the mean value, is incorporated into each boxplot*

*# box.mean.size: numeric value of the losange dot size (in mm). Not considered if box.mean is FALSE #REMOVE*

*# box.whisker.kind: range of the whiskers. Either "no" (no whiskers), or "std" (length of each whisker equal to 1.5 x Inter Quartile Range (IQR)), or "max" (length of the whiskers up or down to the most distant dot)*

*# box.whisker.width: numeric value (from 0 to 1) of the whisker width, with 0 meaning no whiskers and 1 meaning a width equal to the corresponding boxplot width*

*# dot.color: vector of character string. Idem as categ.color but for dots, except that in the possibility (3), the rule "a single color per class of categ1 and a single class of categ1", cannot be respected (each dot can have a different color). If NULL, no dots plotted*

*# dot.categ: optional single character string of a data1 column name (further refered to as categ3). If non NULL, then a legend will be created for the dots, in addition to the legend for the boxes*

*# dot.categ.class.order: optional vector of character strings indicating the order of the classes of categ3. If dot.categ is non NULL and dot.categ.class.order = NULL, classes are displayed in the legend according to the alphabetical order. Ignored if dot.categ is NULL*

*# dot.categ.legend.name: optional character string of the legend title for categ3. If categ.legend.name = NULL, categ3 value is used (name of the column in data1). Write "" if no legend required. Ignored if dot.categ is NULL*

*# dot.tidy: logical. Nice dot spreading? If TRUE, use the geom\_dotplot() function for a nice representation. If FALSE, dots are randomly spread, using the dot.jitter argument (see below)*

*# dot.tidy.bin.nb: positive integer indicating the number of bins (i.e., nb of separations) of the y.lim range. Each dot will then be put in one of the bin, with the size the width of the bin. Not considered if dot.tidy is FALSE*

*# dot.jitter: numeric value (from 0 to 1) of random dot horizontal dispersion, with 0 meaning no dispersion and 1 meaning a dispersion in the corresponding box width interval. Not considered if dot.tidy is TRUE*

*# dot.size: numeric value of dot size (in mm). Not considered if dot.tidy is TRUE*

*# dot.alpha: numeric value (from 0 to 1) of dot transparency (full transparent to full opaque, respectively)*

*# dot.border.size: numeric value of border dot size (in mm). Write zero for no dot border. If dot.tidy is TRUE, value 0 remove the border. Another one leave the border without size control (geom\_doplot() feature)*

*# dot.border.color: single character color string defining the color of the dot border (same color for all the dots, whatever their categories). If dot.border.color = NULL, the border color will be the same as the dot color. A single integer is also accepted instead of a character string, that will be processed by fun\_gg\_palette()*

*# x.lab: a character string or expression for x-axis legend. If NULL, character string of categ1*

*# y.lab: a character string or expression for y-axis legend. If NULL, character string of the y argument*

*# y.lim: 2 numeric values indicating the range of the y-axis*

*# y.log: Either "no" (values in the y argument column of the data1 data frame are not log), "log2" (values in the y argument column of the data1 data frame are log2 transformed) or "log10" (values in the y argument column of the data1 data frame are log10 transformed). BEWARE: do not tranform the data, but just display ticks in a log scale manner. Thus, negative or zero values allowed. BEWARE: not possible to have horizontal boxs with a log axis, due to a bug in ggplot2 (see https://github.com/tidyverse/ggplot2/issues/881)*

*# y.tick.nb: approximate number of desired label values on the y-axis (n argument of the the fun\_scale() function)*

*# y.inter.tick.nb: number of desired secondary ticks between main ticks. Not considered if y.log is other than "no". In that case, play with the y.lim and y.tick.nb arguments*

*# y.include.zero: logical. Does y.lim range include 0? Ok even if y.log = TRUE because y.lim must already be log transformed values*

*# y.top.extra.margin: single proportion (between 0 and 1) indicating if extra margins must be added to y.lim. If different from 0, add the range of the axis \* y.top.extra.margin (e.g., abs(y.lim[2] - y.lim[1]) \* y.top.extra.margin) to the top of y-axis*

*# y.bottom.extra.margin: idem as y.top.extra.margin but to the bottom of y-axis*

*# stat.disp: add the median number above the corresponding box. Either NULL (no number shown), "top" (at the top of the figure region) or "above" (above each box)*

*# stat.disp.mean: logical. Diplay means instead of medians ?*

*# stat.size: numeric value of the stat size (in points). Increase the value to increase text size*

*# stat.dist: numeric value of the stat distance. Increase the value to increase the distance from the box plot*

*# vertical: logical. Vertical boxs? BEWARE: will be automatically set to TRUE if y.log argument is other than "no". Indeed, not possible to have horizontal boxs with a log axis, due to a bug in ggplot2 (see https://github.com/tidyverse/ggplot2/issues/881)*

*# text.size: numeric value of the size of the (1) axis numbers and axis legends, (2) texts in the graphic legend, (3) stats above boxs (in points)*

*# text.angle: integer value of the text angle for the x-axis labels. Positive values for counterclockwise rotation: 0 for horizontal, 90 for vertical, 180 for upside down etc. Negative values for clockwise rotation: 0 for horizontal, -90 for vertical, -180 for upside down etc.*

*# title: character string of the graph title*

*# title.text.size: numeric value of the title size (in points)*

*# classic: logical. Use the classic theme (article like)?*

*# grid: logical. draw horizontal lines in the background to better read the box values? Not considered if classic = FALSE*

*# return: logical. Return the graph parameters?*

*# plot: logical. Plot the graphic? If FALSE and return argument is TRUE, graphical parameters and associated warnings are provided without plotting*

*# add: character string allowing to add more ggplot2 features (dots, lines, themes, etc.). BEWARE: (1) must start with "+" just after the simple or double opening quote (no space, end of line, carriage return, etc., allowed), (2) must finish with ")" just before the simple or double closing quote (no space, end of line, carriage return, etc., allowed) and (3) each function must be preceded by "ggplot2::" (for instance: "ggplot2::coord\_flip()). If the character string contains the "ggplot2::theme" string, then internal ggplot2 theme() and theme\_classic() functions will be inactivated to be reused by add. BEWARE: handle this argument with caution since added functions can create conflicts with the preexisting internal ggplot2 functions*

*# warn.print: logical. Print warnings at the end of the execution? No print if no warning messages. some of the warning messages (those delivered by the internal ggplot2 functions) are not apparent when using the argument plot = FALSE*

*# path.lib: absolute path of the required packages, if not in the default folders*

*# REQUIRED PACKAGES*

*# ggplot2*

*# scales*

*# REQUIRED FUNCTIONS FROM CUTE\_LITTLE\_R\_FUNCTION*

*# fun\_comp\_2d()*

*# fun\_gg\_just()*

*# fun\_gg\_palette()*

*# fun\_name\_change()*

*# fun\_pack()*

*# fun\_check()*

*# fun\_round()*

*# fun\_scale()*

*# RETURN*

*# a boxplot if plot argument is TRUE*

*# a list of the graph info if return argument is TRUE:*

*# $stat: the graphic statistics*

*# $removed.row.nb: which rows have been removed due to NA detection in y and categ columns (NULL if no row removed)*

*# $removed.rows: removed rows containing NA (NULL if no row removed)*

*# $data: the graphic box and dot coordinates*

*# $axes: the x-axis and y-axis info*

*# $warnings: the warning messages. Use cat() for proper display. NULL if no warning. BEWARE: some of the warning messages (those delivered by the internal ggplot2 functions) are not apparent when using the argument plot = FALSE*

*# EXAMPLES*

*### nice representation (1)*

*obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), categ.class.order = list(NULL, c("B", "A")), categ.legend.name = "LEGEND", categ.color = NULL, box.width = 0.3, whisker.width = 0.8, dot.color = "same", dot.jitter = 0.5, dot.size = 3.5, dot.border.size = 0.2, dot.alpha = 0.5, ylim = c(10, 25), y.include.zero = TRUE, stat.disp = "above", stat.size = 4, xlab = "GROUP", ylab = "VALUE", text.size = 12, title = "GRAPH1", title.text.size = 8, text.angle = 0, classic = TRUE, grid = TRUE)*

*### nice representation (2)*

*set.seed(1) ; obs1 <- data.frame(Time = c(rnorm(24, 0), rnorm(24, -10), rnorm(24, 10), rnorm(24, 20)), Group1 = rep(c("CAT", "DOG"), times = 48), Group2 = rep(c("A", "B", "C", "D"), each = 24)) ; set.seed(NULL) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), categ.class.order = list(NULL, c("B", "A", "D", "C")), categ.legend.name = "LEGEND", categ.color = NULL, box.width = 0.8, dot.color = "grey50", dot.tidy = TRUE, dot.bin.nb = 60, dot.size = 3.5, dot.border.size = 0.2, dot.alpha = 0.5, ylim= c(-20, 30), stat.disp = "above", stat.size = 4, stat.dist = 1, xlab = "GROUP", ylab = "VALUE", vertical = FALSE, text.size = 12, title = "GRAPH1", title.text.size = 8, text.angle = 45, classic = FALSE)*

*### separate boxes. Simple example*

*set.seed(1) ; obs1 <- data.frame(Time = c(rnorm(10), rnorm(10) + 2), Group1 = rep(c("G", "H"), each = 10)) ;*

*fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = "Group1")*

*### separate boxes. Changing the order of the boxes*

*fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = "Group1", categ.class.order = list(c("H", "G")))*

*### separate boxs. Example (1) of modification of box color using a single value*

*obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = "Group1", categ.color = "white")*

*### separate boxs. Example (2) of modification of box color using one value par class of categ2*

*obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = "Group1", categ.color = c("coral", "lightblue"))*

*### separate boxs. Example (3) of modification of box color using the box.color data frame column, with respect of the correspondence between categ2 and box.color columns*

*obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), box.color = rep(c("coral", "lightblue"), time = 10)) ; obs1 ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = "Group1", categ.color = obs1$box.color)*

*### separate boxs. Example (1) of modification of dot color, using the same dot color as the corresponding box*

*obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = "Group1", dot.color = "same")*

*### separate boxs. Example (2) of modification of dot color, using a single color for all the dots*

*obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = "Group1", dot.color = "green")*

*### separate boxs. Example (3) of modification of dot color, using one value par class of categ2*

*obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = "Group1", dot.color = c("green", "brown"))*

*### separate boxs. Example (4) of modification of dot color, using different colors for each dot*

*obs1 <- data.frame(Time = 1:10, Group1 = rep(c("G", "H"), times = 5)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = "Group1", dot.color = hsv(h = (1:nrow(obs1)) / nrow(obs1)))*

*### grouped boxs. Simple example*

*set.seed(1) ; obs1 <- data.frame(Time = c(rnorm(20), rnorm(20) + 2), Group1 = rep(c("G", "H"), each = 10), Group2 = rep(c("A", "B"), time = 10)) ;*

*fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"))*

*### grouped boxs. More grouped boxs*

*obs1 <- data.frame(Time = 1:24, Group1 = rep(c("G", "H"), times = 12), Group2 = rep(c("A", "B", "C", "D"), each = 6)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"))*

*### grouped boxs. Example (1) of modification of box color, using a single value*

*obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), categ.color = "white")*

*### grouped boxs. Example (2) of modification of box color, using one value par class of categ2*

*obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), categ.color = c("coral", "lightblue"))*

*### grouped boxs. Example (3) of modification of box color, using one value per line of obs1, with respect of the correspondence between categ2 and box.color columns*

*obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10), box.color = rep(c("coral", "lightblue"), each = 10)) ; obs1 ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), categ.color = obs1$box.color)*

*### grouped boxs. Example (1) of modification of dot color, using the same dot color as the corresponding box*

*obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = "same")*

*### grouped boxs. Example (2) of modification of dot color, using a single color for all the dots*

*obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = "green")*

*### grouped boxs. Example (3) of modification of dot color, using one value par class of categ2*

*obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = c("green", "brown"))*

*### grouped boxs. Example (4) of modification of dot color, using different colors for each dot*

*obs1 <- data.frame(Time = 1:10, Group1 = rep(c("G", "H"), times = 5), Group2 = rep(c("A", "B"), each = 5)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = hsv(h = (1:nrow(obs1)) / nrow(obs1)))*

*### no dots (y.include.zero set to TRUE to see the lowest box):*

*obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = NULL, y.include.zero = TRUE)*

*### box width. Example (1) with box.width = 0.25 -> three times more space between single boxs than the box width (y.include.zero set to TRUE to see the lowest box)*

*obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), each = 500)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = "Group1", dot.color = NULL, y.include.zero = TRUE, box.width = 0.25)*

*### box width. Example (2) with box.width = 1, no space between single boxs*

*obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), each = 500)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = "Group1", dot.color = NULL, y.include.zero = TRUE, box.width = 1)*

*### box width. Example (3) with box.width = 0.25 -> three times more space between sets of grouped boxs than the set width*

*obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), times = 500), Group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = NULL, y.include.zero = TRUE, box.width = 0.25)*

*### box width. Example (4) with box.width = 0 -> no space between sets of grouped boxs*

*obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), times = 500), Group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = NULL, y.include.zero = TRUE, box.width = 1)*

*### whisker width. Example (1) with whisker.width = 1 -> whiskers have the width of the corresponding box*

*obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), times = 500), Group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = NULL, whisker.width = 1)*

*### whisker width. Example (2) error boxs with no whiskers*

*obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), times = 500), Group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = NULL, whisker.width = 0)*

*### tidy dot distribution. Example (1)*

*obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), times = 500), Group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = "same", dot.tidy = TRUE, dot.bin.nb = 100)*

*### tidy dot distribution. Example (2) reducing the dot size with dot.bin.nb*

*obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), times = 500), Group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = "same", dot.tidy = TRUE, dot.bin.nb = 150)*

*### dot jitter. Example (1)*

*obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), times = 500), Group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = "same", dot.tidy = FALSE, dot.jitter = 1, dot.size = 2)*

*### dot jitter. Example (2) with dot.jitter = 1 -> dispersion around the corresponding box width*

*obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), times = 500), Group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = "grey", dot.size = 3, dot.alpha = 1, dot.jitter = 1)*

*### dot jitter. Example (3) with no dispersion*

*obs1 <- data.frame(Time = 1:100, Group1 = rep(c("G", "H"), times = 50), Group2 = rep(LETTERS[1:5], each = 20)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = "grey", dot.size = 3, dot.alpha = 1, dot.jitter = 0)*

*### dot size, dot border size and dot transparency*

*obs1 <- data.frame(Time = 1:100, Group1 = rep(c("G", "H"), times = 50), Group2 = rep(LETTERS[1:5], each = 20)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = "grey", dot.size = 4, dot.border.size = 0, dot.alpha = 0.6)*

*### y-axis limits. Example (1)*

*obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), ylim = c(-1, 25))*

*### y-axis limits. Example (2) showing that order matters in ylim argument*

*obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), ylim = c(25, -1))*

*### log scale. Example (1). BEWARE: y column must be log, otherwise incoherent scale (see below warning message with the return argument)*

*obs1 <- data.frame(Time = log10((1:20) \* 100), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), ylog = "log10")*

*### log scale. Example (2). BEWARE: values of the ylim must be in the corresponding log*

*obs1 <- data.frame(Time = log10((1:20) \* 100), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), ylog = "log10", ylim = c(1,4))*

*### tick number. Example (1)*

*obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), y.tick.nb = 10)*

*### tick number. Example (2) using a log2 scale*

*obs1 <- data.frame(Time = log2((1:20) \* 100), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), ylog = "log2", y.tick.nb = 10, ylim = c(1, 16))*

*### tick number. Example (3) using a log10 scale*

*obs1 <- data.frame(Time = log10((1:20) \* 100), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), ylog = "log10", y.tick.nb = 10)*

*### tick number. Example (4) using a log10 scale: the reverse y-axis correctly deal with log10 scale*

*obs1 <- data.frame(Time = log10((1:20) \* 100), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), ylog = "log10", y.tick.nb = 10, ylim = c(4, 1))*

*### secondary tick number. Example (1)*

*obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), y.inter.tick.nb = 2)*

*### secondary ticks. Example (2) not for log2 and log10 scales (see below warning message with the return argument)*

*obs1 <- data.frame(Time = log10((1:20) \* 100), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), ylog = "log10", y.inter.tick.nb = 2)*

*### include zero in the y-axis*

*obs1 <- data.frame(Time = (1:20), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), y.include.zero = TRUE)*

*### extra margins. To avoid dot cuts*

*obs1 <- data.frame(Time = (1:20), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), y.top.extra.margin = 0.25, y.bottom.extra.margin = 0.25)*

*### mean diplay. Example (1) at the top of the plot region*

*obs1 <- data.frame(Time = (1:20), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), y.top.extra.margin = 0.1, stat.disp = "top", stat.size = 4, stat.dist = 2)*

*### mean diplay. Example (2) above boxs*

*obs1 <- data.frame(Time = (1:20), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), y.top.extra.margin = 0.1, stat.disp = "above", stat.size = 4, stat.dist = 2)*

*### box orientation. Example (1) without log scale, showing that the other arguments are still operational*

*obs1 <- data.frame(Time = (1:20), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), y.tick.nb = 10, y.inter.tick.nb = 2, y.include.zero = TRUE, vertical = FALSE)*

*### box orientation. Example (2) with log scale. Horizontal orientation is blocked with log2 and log10 scales because of a bug in ggplot2 (https://github.com/tidyverse/ggplot2/issues/881)*

*obs1 <- data.frame(Time = log10((1:20) \* 100), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), ylog = "log10", vertical = FALSE)*

*### classic representation (use grid = TRUE to display the background lines of the y axis ticks)*

*obs1 <- data.frame(Time = (1:20), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), classic = TRUE, grid = FALSE)*

*### graphic info. Example (1)*

*obs1 <- data.frame(Time = log10((1:20) \* 100), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), return = TRUE)*

*### graphic info. Example (2) of assignation and warning message display*

*obs1 <- data.frame(Time = log10((1:20) \* 100), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; warn <- fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), ylog = "log10", return = TRUE) ; cat(warn$warnings)*

*### add ggplot2 functions*

*obs1 <- data.frame(Time = log10((1:20) \* 100), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), add = "+ggplot2::theme\_classic()")*

*### all the arguments*

*obs1 <- data.frame(x = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "x", categ = c("Group1", "Group2"), categ.class.order = list(NULL, c("B", "A")), categ.legend.name = "", categ.color = c("red", "blue"), box.width = 0.25, whisker.width = 0.8, dot.color = "grey", dot.tidy = FALSE, dot.bin.nb = 30, dot.jitter = 1, dot.size = 4, dot.border.size = 0, dot.alpha = 1, ylim = c(0, 25), ylog = "no", y.tick.nb = NULL, y.inter.tick.nb = NULL, y.include.zero = FALSE, y.top.extra.margin = 0.05, y.bottom.extra.margin = 0, stat.disp = "above", stat.size = 4, stat.dist = 2, xlab = "GROUP", ylab = "VALUE", vertical = FALSE, text.size = 12, title = "", title.text.size = 8, text.angle = 45, classic = TRUE, grid = TRUE, return = TRUE, plot = TRUE, add = NULL, warn.print = TRUE, path.lib = NULL)*

*# problem of warning message*

*# error with dot.tidy = TRUE*